Systems programming

5 – Processes introduction

MEEC LEEC MEAer LEAer MEIC-A
João Nuno Silva

TÉCNICO
LISBOA

Bibliography

- Beej's Guide to Interprocess Communication, Brian Hall
  - Chapters 2
- The Linux Programming Interface, Michael Kerrisk
  - Chapter 24

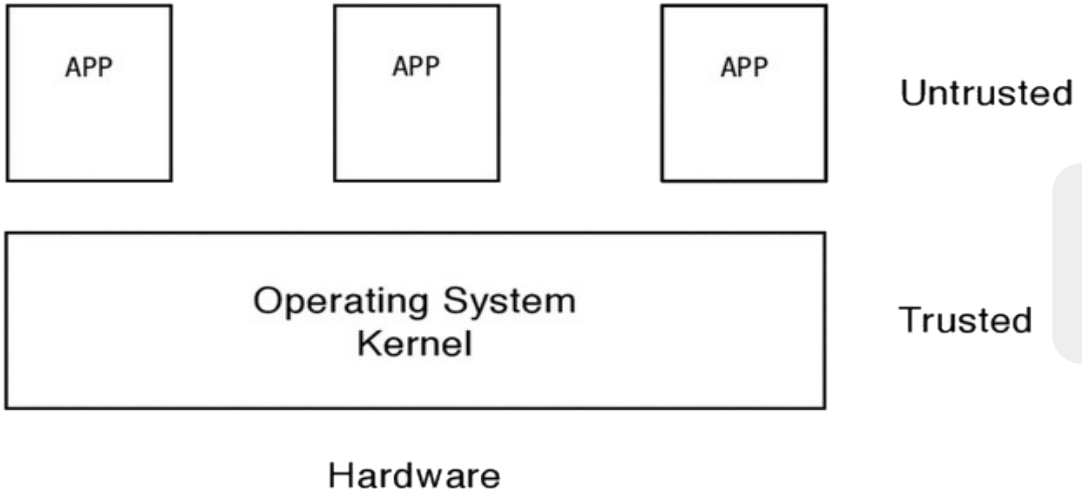
Processes

Protection

- Operating Systems central role
  - isolation of misbehaving applications
- Fundamental to other OS goals
  - Reliability
  - Security
  - Privacy
  - fairness
- OS kernel
  - lowest level SW running on the system
  - implements protection

Protection

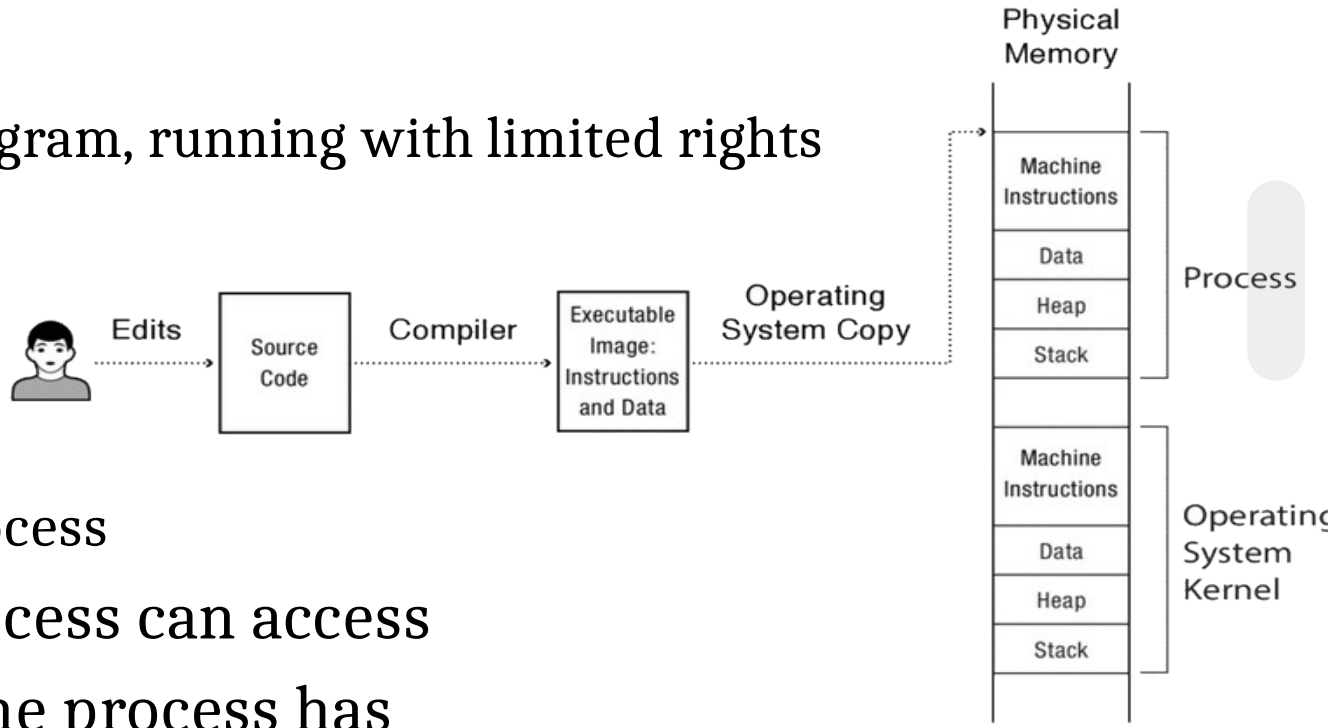
- Applications
  - untrusted



- Process
  - execution of application program with restricted rights
  - Needs permission
    - from OS kernel
    - to access resources (memory from other processes, I/O. ...)

Process Abstraction

- Process:
  - an instance of a program, running with limited rights



- Address space:
  - set of rights of a process
- Memory that the process can access
- Other permissions the process has
  - e.g., which system calls it can make, what files it can access)

UNIX Process Management

- Basic API to manage processes
  - Creation
  - Termination
  - Reception of return code
    - Remember C exit function

UNIX Process Management

- UNIX fork
  - system call to create a copy of the current process, and start it running
  - No arguments!
- UNIX exit
  - system call to terminate a process
- UNIX wait
  - system call to wait for a process to finish
- UNIX signal
  - system call to send a notification to another process

Unix process creation

- **Existing process (father)**
  - Calls fork system call
  - Waits for start of new project
  - Continues execution
- **New process (son)**
  - Is created
  - Continues execution

- **Kernel**
  - Receives request
  - Replicates calling process
  - Starts new process
  - Notifies existing process

Unix process creation

- To start a new process call:
  - `#include <unistd.h>`
  - `pid_t fork();`
- A new process is created

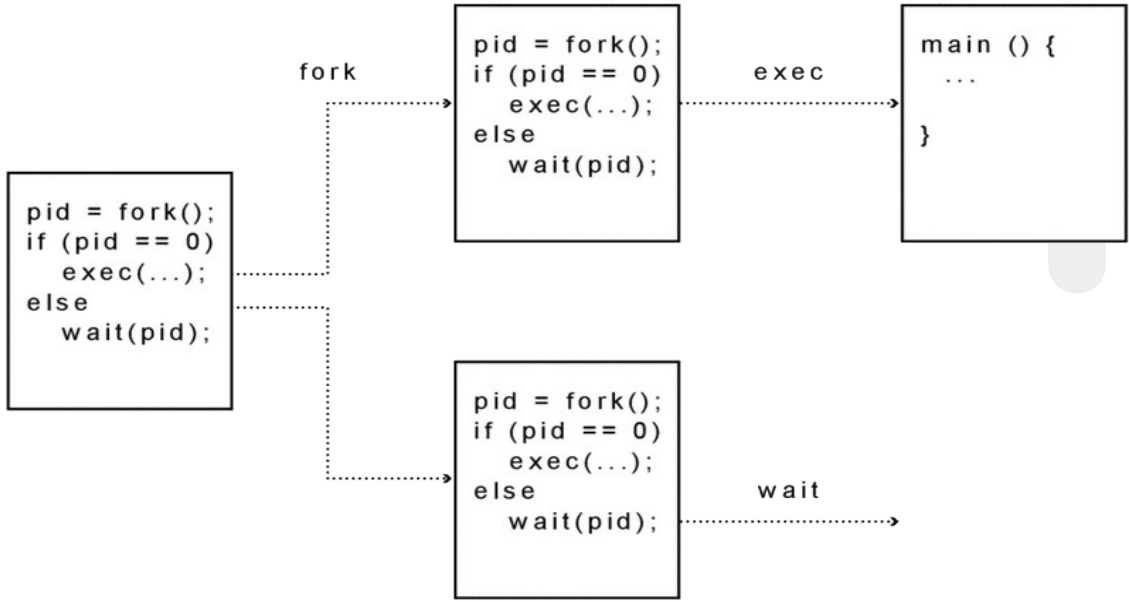
- **`Man fork`**

Unix `fork`

- The new instruction to be executed is the one following the fork
  - On the father and son !!!!
- All variables are duplicated with the same value
  - After the fork all changes in values are local to each process.
- Fork is a system call that return two different values:
  - In the son return 0
  - In the father return the son PID

Unix fork

```
int new_pid = fork();
if (new_pid == 0) {
  printf("child PID #%d\n",
                 getpid());
  exit(0);
} else {
  printf("Parent PID %d\n",
                 child_pid);
  exit(0);
}
```



Unix Process creation policies

- Execution mode
  - Father and son execute concurrently
- Memory management
  - Son gets a NON SHARED copy of father memory
- Resource sharing (files, ...)
  - Father and son share all resources
- Child inherits
  - Copy of most information
  - Copy of Memory space
    - Variables
    - Allocated memory
  - Code
  - Opened files
    - pipes, fifos, sockets

Process termination

- When a process executes the exit(int) function:
  - The calling process is terminated "immediately".
  - Any open file belonging to the process are closed;
  - Process's parent is sent a SIGCHLD signal.

- **Man 2 exit / man 3 exit**
  - What happens to the return code?

Reception of return code

- If parent needs the child return code:
  - It must wait for its dead:
  - Call **`wait`** /**`waitpid`** function

pid_t waitpid(pid_t pid, int *status, int options);

  - Process waits for a specifi child termination.
  - 1st argument – ID of child (-1 any process)
  - 2nd argument – child status
  - 3rd argument
    - WNOHANG: return immediately if no child has exited
    - WUNTRACED: also return if a child has stopped

Reception of return code

- **WIFEXITED(status)**
  - returns true if the child terminated normally, exit(3) _exit(2) or by returning from main().
- **WEXITSTATUS(status)**
  - returns the exit status of the child.
- **WIFSIGNALED(status)**
  - returns true if the child process was terminated by a signal.
- **WTERMSIG(status)**
  - returns the number of the signal that caused the child process to terminate.
- **WIFSTOPPED(status)**
  - returns true if the child process was stopped by delivery of a signal
- **WSTOPSIG(status)**
  - returns the number of the signal which caused the child to stop.

Next on PSIS

- Pipes